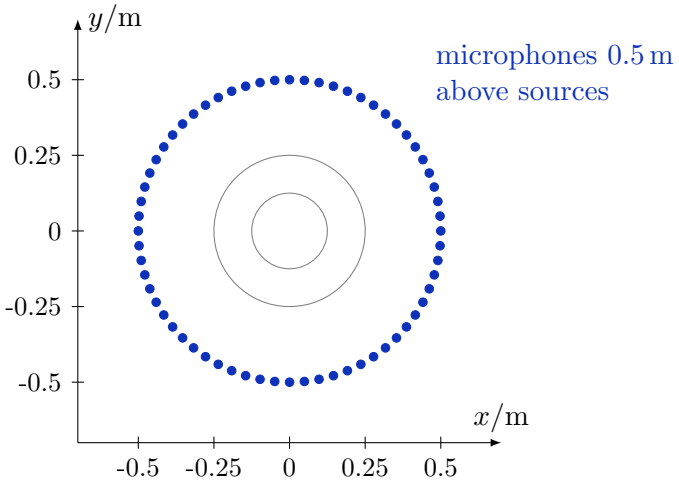
Microphone Array Benchmark b11: Rotating Point Sources

Gert Herold
2019-05-02*

This benchmark case is part of a collection of synthetic and experimental array data sets compiled by the microphone array community for testing and comparing different microphone array methods. It consists of two simulated data sets (subcases a and b) featuring rotating point sources. Sound pressure time data are generated at 64 microphones, which are arranged on a circle with a diameter of 1 m. The simulation setup is chosen to be similar to the experimental setup used by Zenger et al. [1]. Simulations were done using the Python package Acoular [2], publicly available under www.acoular.org.

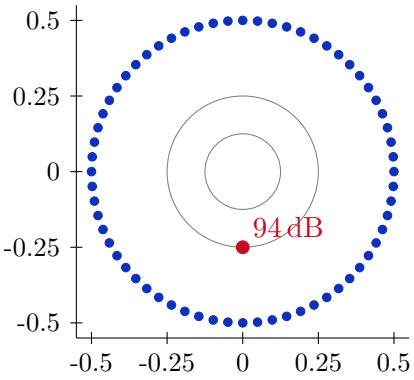
Simulation setup



- 64 microphones in one ring, $d_{\text{array}} = 1\,\text{m}$, equidistant arrangement (clockwise)
- first channel at $(r, \varphi) = (0.5\,\text{m}, 90°)/(x, y) = (0, 0.5)\text{m}$
- distance of array plane to source plane $z = 0.5\,\text{m}$
- array center aligned with axis of rotation of the sources
- sampling rate $f_{\text{sample}} = 48\,\text{kHz}$, measurement time $t_{\text{meas}} = 10\,\text{s}$
- no flow, no additional noise, speed of sound is $c = 343.0\,\frac{\text{m}}{\text{s}}$
- tacho signal (1 trigger / revolution)

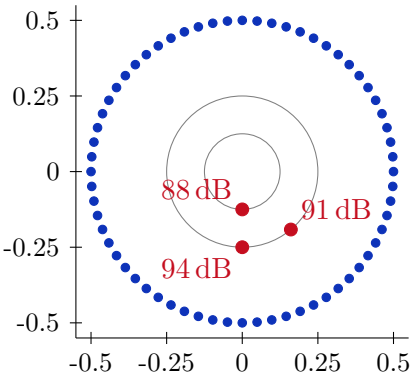
*Document history: V.1: 2017-07-28, V.2: 2018-09-28. Data sets remain unchanged.
Author contact: gert.herold@tu-berlin.de

Subcase a – 1 point source

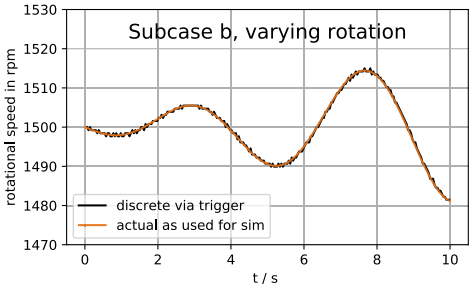


- 1 rotating point source at $r = 0.25\,\mathrm{m}$
- clock-wise rotation with a rate of 1500 rpm (constant)
- emitting white noise signal[1]

Subcase b – 3 point sources



- 3 point sources
- all rotating clock-wise at the same angular rate
- $r_1 = r_2 = 0.25\,\mathrm{m}, r_3 = 0.125\,\mathrm{m}$
- $\varphi_1 = \varphi_3 = \varphi_2 - 40°$
- slightly varying overall rotational rate (≈ 1500 rpm)
- emitting uncorrelated white noise[1]
- $L_{p,1} = L_{p,2} + 3\,\mathrm{dB} = L_{p,3} + 6\,\mathrm{dB}$



[1]Levels displayed in the figures correspond to the sound pressure level at 1 m distance from the respective point source.

Challenge

- Determine the position of all sources (r, φ) relative to the array at the trigger instant.
- Determine the 1/3 octave band spectra for the detected sources.

List of files

The format of the benchmark data files follows the *Array Methods HDF5 File Definitions* [3]. Python code for simulating the data is shown at the end of this document.

- subcase a
  - `b11aTimeSeries.h5` – time series
  - `b11aCsmEss.h5` – cross spectral matrix
  - `b11aTimeSeriesOpt.h5` – tacho signal (1 trigger per revolution, 5 V peak)
- subcase b
  - `b11bTimeSeries.h5` – time series
  - `b11bCsmEss.h5` – cross spectral matrix
  - `b11bTimeSeriesOpt.h5` – tacho signal (1 trigger per revolution, 5 V peak)

The additional time series files containing the tacho signals are structured similarly to the files with the microphone data: Under the `TachoData` key, the `tachoDataV` data set features a `sampleCount`×1 array. The sample rate of the tacho signal is the same as for the microphone data (48 000 Hz). Each revolution is marked by a value of 5.0, all other entries are zero. The first trigger is set to occur after a half-revolution. The CSM files are included for meeting the requirements for completeness of the data set [3], but are not actually needed for evaluation.

References

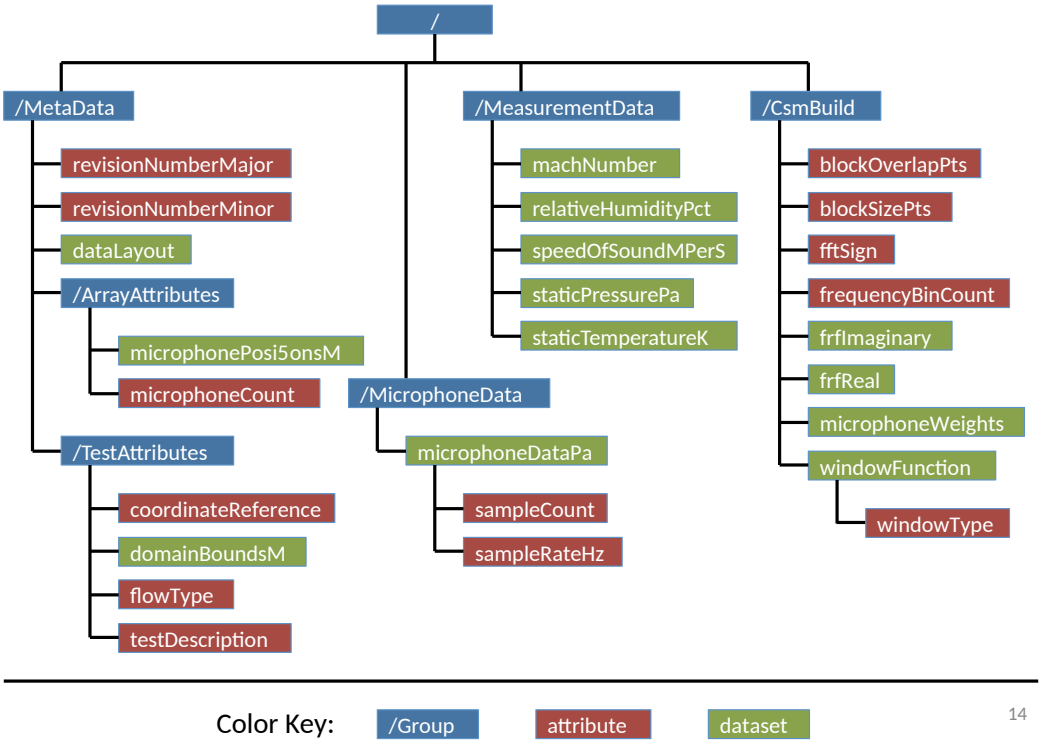
[1] F. J. Zenger, G. Herold, S. Becker, and E. Sarradj, "Sound source localization on an axial fan at different operating points", *Experiments in Fluids*, 57(8), 1–10, 2016. doi:10.1007/s00348-016-2223-8

[2] E. Sarradj, G. Herold, "A Python framework for microphone array data processing", *Applied Acoustics*, 116, 50–58, 2017. doi:10.1016/j.apacoust.2016.09.015

[3] Array Methods HDF5 File Definitions, Revision 2.4
https://www-docs.b-tu.de/fg-akustik/public/veroeffentlichungen/ArrayMethodsFileFormatsR2P4Release.pdf
www.b-tu.de/fg-akustik/lehre/aktuelles/arraybenchmark, Last accessed: 2019-05-02.

Excerpt of used HDF5 file format documentation [3]

Array Methods HDF5 Time Series File Layout



14

Array Methods HDF5 Time Series File Definitions

- /MicrophoneData
  - microphoneDataPa
    - microphone time series data
    - size: sampleCount x microphoneCount
    - units: Pa
    - format: floating point
    - chunk size: : sampleCount x 1
    - note: each chunk is a single microphone time series
    - note: units are selected to account for microphone sensitivity and any amplifier settings; please remember that some transducers have a negative voltage change for a positive pressure change, and account for this in the conversion of the data
    - sampleCount
      - number of samples in a single microphone time series
      - format: 32-bit signed integer
    - sampleRateHz
      - rate of data acquisition
      - units: samples per second
      - format: floating point

18

Python code to generate the data

```
# -*- coding: utf-8 -*-
"""
Created on 2019-04-30, License: CC BY 4.0

@author: Gert Herold

Simulate rotating data incl. trigger signal
"""

from acoular import MicGeom, WNoiseGenerator, Trajectory, MovingPointSource, \
SourceMixer, TimeCache, PowerSpectra
from numpy import pi, linspace, array, cos, sin, sqrt, int32, arange, \
zeros, less
from scipy.signal import argrelextrema

import h5py
from AIAAwrite import AIAAwrite

##############################################################################

# position of sources
z1 = z2 = z3 = 0.5 # m
r1 = r2 = 0.25
r3 = 0.125

# starting angle -- NOT angle at trigger instant
phi1 = phi3 = 90./180.*pi
phi2 = 130./180.*pi

# rotational rate, anti-clockwise
rpm = -1500 # rpm
rps = rpm/60.

delta_t = 1./abs(rps)/16.0 # ca. 16 spline nodes per revolution

# measurement time
tmax = 10 # s

# sampling rate
sfreq = 48000 # Hz
nsamples = tmax * sfreq

# microphone geometry, clock-wise arrangement, starting at phi0 = 90
rmics = 0.5
phis = -1 * linspace(0, 2*pi, 64, endpoint=False) + pi/2

m = MicGeom()
mpos = array( [ (rmics*cos(phi), rmics*sin(phi), 0) for phi in phis ] ).T
mpos[abs(mpos)<1e-15] = 0.0 # set zeros to zero
m.mpos_tot = mpos

# source signals (white noise)
n1 = WNoiseGenerator(sample_freq=sfreq, numsamples=nsamples, seed = 1, rms=1.0)
n2 = WNoiseGenerator(sample_freq=sfreq, numsamples=nsamples, seed = 2, rms=sqrt(0.5))
n3 = WNoiseGenerator(sample_freq=sfreq, numsamples=nsamples, seed = 3, rms=0.5)

##############################################################################

# function to export trigger signal
def write_trigger(trigger, case = 'default'):
    # open h5 file and prepare
    h5f = h5py.File('%sTimeSeriesOpt.h5'%case,'w')
    TachoData = h5f.create_group('TachoData')
    tachoDataV = TachoData.create_dataset('tachoDataV',
                                          shape=(nsamples,1),
                                          chunks=(nsamples,1))
    tachoDataV[:,:] = trigger.reshape((nsamples,1))
    tachoDataV.attrs.create('sampleCount', nsamples, dtype=int32)
    tachoDataV.attrs.create('sampleRateHz', sfreq)
    tachoDataV.attrs.create('units','V'.encode(encoding='utf-8'))
    h5f.close()

##############################################################################

```

```
### subcase a ###

# trajectory of source
tr0 = Trajectory()

for t in arange(0, tmax*1.001, delta_t):
    phi = t * rps * 2 * pi #angle
    # define points for trajectory spline
    tr0.points[t] = (r1*cos(phi+phi1), r1*sin(phi+phi1), z1)

# point source
p0 = MovingPointSource(signal=n1, mics=m, trajectory=tr0)

samples_per_rotation = int(abs(sfreq/rps))
trigger = zeros((nsamples,1))
ind = samples_per_rotation//2 # trigger when source is at 6 o'clock
while ind < nsamples:
    trigger[ind] = 5.0
    ind += samples_per_rotation

# time data here only consists of source 1
ta = TimeCache(source=p0)
fa = PowerSpectra(time_data=ta, precision='complex64',
                  window='Hanning', overlap='50%', block_size=1024)

AIAAwrite(m,fa,ta,case="b11a")
write_trigger(trigger, case="b11a")

##############################################################################

### subcase b ###

# jitter
d_rpm = 1.5
d_rps = d_rpm/60.
f_jitter = 0.2 # Hz

# trajectories
tr1 = Trajectory()
tr2 = Trajectory()
tr3 = Trajectory()

for t in arange(0, tmax*1.001, delta_t):
    jitter = d_rps * sin(2*pi*f_jitter*t)
    phi = t * (rps+jitter) * 2 * pi #angle
    # define points for trajectory spline
    tr1.points[t] = (r1*cos(phi+phi1), r1*sin(phi+phi1), z1)
    tr2.points[t] = (r2*cos(phi+phi2), r2*sin(phi+phi2), z2)
    tr3.points[t] = (r3*cos(phi+phi3), r3*sin(phi+phi3), z3)

# point sources
p1 = MovingPointSource(signal=n1, mics=m, trajectory=tr1)
p2 = MovingPointSource(signal=n2, mics=m, trajectory=tr2)
p3 = MovingPointSource(signal=n3, mics=m, trajectory=tr3)

# mix 3 sources
p = SourceMixer(sources = [p1,p2,p3])

# generate trigger signal -> trigger when sin(phi) = 0 and cos(phi) < 0
t = linspace(0,tmax,nsamples,endpoint=False)
jitter = d_rps * sin(2*pi*f_jitter*t)
phi = t * (rps+jitter) * 2 * pi #angle
sinphi = sin(phi)
cosphi = cos(phi)
# find zeros in sin phi where cos phi is negative
indmin = argrelextrema(sinphi**2, less) # sin(phi)=0 where sin^2(phi) has minima
trigger = zeros((nsamples,))
trigger[indmin] = 5.0
trigger *= cosphi<0 # only keep minima where cos(phi)<0
trigger[0] = 0
trigger[-1] = 0

tb = TimeCache(source=p)
fb = PowerSpectra(time_data=tb,precision='complex64',
                  window='Hanning', overlap='50%', block_size=1024)

AIAAwrite(m,fb,tb,case="b11b")
write_trigger(trigger, case="b11b")

##############################################################################
```